computers

MDPI

Article

*k*NN Prototyping Schemes for Embedded Human Activity Recognition with Online Learning †

Paulo J. S. Ferreira *,‡, João M. P. Cardoso ‡ and João Mendes-Moreira ‡

INESC TEC, Faculty of Engineering, University of Porto, 4200-465 Porto, Portugal; jmpc@fe.up.pt (J.M.P.C.); jmoreira@fe.up.pt (J.M.-M.)

* Correspondence: up201305617@fe.up.pt

† This paper is an extended version of our paper published in Ferreira P.J.S., Magalhães R.M.C., Garcia K.D., Cardoso J.M.P., Mendes-Moreira J. An Efficient Scheme for Prototyping kNN in the Context of Real-Time Human Activity Recognition. In Proceedings of the 20th International Conference, Manchester, UK, 14–16 November 2019.

‡ These authors contributed equally to this work.

Received: 16 October 2020; Accepted: 30 November 2020; Published: 3 December 2020

Abstract: The *k*NN machine learning method is widely used as a classifier in Human Activity Recognition (HAR) systems. Although the *k*NN algorithm works similarly both online and in offline mode, the use of all training instances is much more critical online than offline due to time and memory restrictions in the online mode. Some methods propose decreasing the high computational costs of *k*NN by focusing, e.g., on approximate *k*NN solutions such as the ones relying on Locality-Sensitive Hashing (LSH). However, embedded *k*NN implementations also need to address the target device's memory constraints, especially as the use of online classification needs to cope with those constraints to be practical. This paper discusses online approaches to reduce the number of training instances stored in the *k*NN search space. To address practical implementations of HAR systems using *k*NN, this paper presents simple, energy/computationally efficient, and real-time feasible schemes to maintain at runtime a maximum number of training instances stored by *k*NN. The proposed schemes include policies for substituting the training instances, maintaining the search space to a maximum size. Experiments in the context of HAR datasets show the efficiency of our best schemes.

Keywords: k-nearest neighbor; classification; *k*NN prototyping; Human Activity Recognition (HAR); performance

1. Introduction

Human Activity Recognition (HAR) aims to automatically recognize activities performed by humans through the analyses of sensing data [1]. In recent years, HAR has been widely used in many applications (e.g., [2–4], and [1]), which include health, elderly care and well-being [5], tracking [6,7], and mobile security [8]. Most of the these approaches have been developed using Machine Learning methods, including Decision Trees (see, e.g., [9–11]), Naive Bayes (see e.g., [12–14]), SVM [15], *k*NN [16], and deep learning techniques (see, e.g., [17–21]). However, most of these studies only consider offline learning and, thus, do not address the update of the model after the training [22]. Offline learning algorithms assume that the training data is available before the training phase. Once the dataset is used to train the algorithm, a model is obtained and cannot be trained any further. On the other hand, an online learning algorithm views the data as a stream continuously arriving over time, where the model can be updated and adapt to changes [22].

Several authors have already shown the relevance of using online learning approaches to deal with HAR [22–24], due to the advantage of updating the model with data collected by the user of the system. Among the HAR approaches, *k*NN is both one of the simplest classifiers and one that obtains better predictive performance [24]. However, the storage and computational requirements of *k*NN may prevent its use when targeting wearable devices and smartphones, and specially when considering online learning. The *k*NN classification accuracy and its simplicity to implement online learning are the two main reasons to enhance the prototyping of *k*NN-based HAR systems when targeting online learning. Moreover, prototyping enhancements for *k*NN-based HAR systems can be easily extended to other application domains.

*k*NN is a lazy learning algorithm, because it does not have a learning phase and instead it "memorizes" the training dataset. A lazy learner simply stores the training data, and only when it sees a test instance starts an instance-based approach to classify the test instance based on its nearest stored training instances (i.e., the k nearest neighbors). The larger the training set is, the larger the *k*NN predictive computational cost is. In this paper, we propose extensions to make *k*NN more competitive to online learning in terms of predictive computational cost, its main weakness when used online, without loosing its predictive performance.

We focus on the prototyping of *k*NN-based HAR systems with online learning and consequently needing to be aware of storage constraints. Specifically, we propose various substitution schemes for *k*NN Online Learning approaches, when it is necessary to dynamically update the learning set for each new instance and when there are practical prototyping constraints imposing a maximum number of instances stored. Each substitution scheme is a strategy to keep a limited number of training instances and to guarantee that each class activity is equally represented. Our main goal is to provide an alternative scheme to more simplistic versions of *k*NN, such as the ones that always substitute the oldest instance when the instance limit is reached. This simplistic substitution policy can lead to situations where a given class runs out of training instances. Compared to those versions of *k*NN, our proposed schemes enhance *k*NN prototypes and improve their accuracy with small increases in energy consumption. We note that this paper is an extended version of our previous work [25], where we have presented a first approach on using substitution schemes for both updating the training instances and maintaining a maximum number of instances stored by standard *k*NN implementations.

This paper makes the following contributions:

- Use of various substitution schemes in the context of both standard *k*NN implementations and an LSH-based *k*NN implementation;
- Use of two HAR datasets (*WISDM* [26] and *PAMAP2* [7,27]) to provide scenarios representative of real situations, and to evaluate the accuracy of the proposed substitution schemes;
- Comparison, in terms of accuracy, of the best substitution scheme with other *k*NN methods, namely *k*NNwithPAW [28], *k*NNwithPAWwithADWIN [29], *k*NN default, and *k*NN + LSH [30,31];
- Analysis, in terms of energy consumption and execution time, of the various approaches considered.

This paper is organized as follows. Section 2 presents the related work regarding *k*NN prototyping, especially in the context of embedded systems and possibly considering online learning. In Section 3 we describe the proposed substitution schemes. Section 4 describes the experimental setup used and Section 5 shows experimental results. Finally, Section 6 presents our main conclusions and future work plans.

2. Related Work

Several authors have addressed the prototyping of *k*NN. When considering Prototype Reduction [32], one crucial aspect is the reduction of stored training instances. Two approaches for Prototype Reduction have been typically considered: Prototype Selection [33] and Prototype Generation [34]. Prototype Generation is usually even more computing demanding than Prototype

Selection. This paper is focused on Prototype Selection, specifically to maintain a maximum of training instances stored, which can be thought of as Prototype Substitution when considering online learning. One of the first approaches for Prototype Selection is the Condensed Nearest Neighbor technique [35]. However, the technique is runtime demanding as it decides to store the training instances according to a global and computing demanding view.

Although Prototype Reduction using feature selection [36] is also crucial for HAR systems, at the moment, we consider this as a step when defining the training instances to be stored and deployed and not an online task. The runtime inclusion of feature selection is a complex problem and requires further research to identify its feasibility and associated cost. Feature selection is thus orthogonal to the work presented in this paper, and it can improve execution time and energy consumption, and the overall performance of the classifier.

Some authors have addressed the reduction of the computing demands of *k*NN by reducing the number of distances necessary to calculate during classification. Examples of this are the use of approximated similarity searching (ASS) techniques, such as the ones clustering the training instances [37] and/or using hashing-based approaches [38,39], namely Local Sensitive Hashing (LSH) [40]. However, we note that the use of ASS techniques is orthogonal to the approaches presented in this paper as we focus on the substitution of the stored instances when dynamically dealing with new training instances (and to maintain a maximum number of training instances in the prototype). We also present the impact of our proposed substitution schemes in an LSH-based *k*NN [41].

The following approaches use different methods to select and remove instances from the *k*NN memory of training instances.

The ADWIN (ADaptive sliding WINdowing) method [29] monitors statistical changes in a sliding window. The window stores all the instances since the last detected change. The window is partitioned into two sub-windows of various sizes until the difference of their average error exceeds a threshold. A change is detected based on the sub-windows' size and a confidence parameter, and the older window is dropped. The algorithm automatically grows the window when no change is apparent and shrinks it when data changes.

The method PAW (Probabilistic Approximate Window) [28] stores in a sliding window only the most recent and relevant instances according to a probabilistic metric. To decide which instance to maintain or discard, the algorithm randomly selects the instances kept in the window, making a mix of recent and older instances.

In [28], ADWIN is coupled with *k*NN with PAW. In this case, ADWIN is used to keep only the data related to the most recent concept of the stream, and the rest of instances are discarded.

A SAM (Self Adjusting Memory) [42] is coupled with the *k*NN classifier and can adapt to concept drift by explicitly separating current and past information using two memory structures, one based on short-term-memory and the other on long-term-memory. The short-term-memory contains data of the current streaming concept, and the long-term-memory maintains knowledge (old models) of past concepts.

The SWC (Sliding Window Clustering) method [23], instead of storing all instances that fit in a sliding window (for representing both old and current concepts), stores compressed information about concepts and instances close to the uncertainty border of each class. The clusters are compressed stable concepts, and the instances are possible drifts of these concepts.

Besides the focus on dealing with the training instances and their memorization, there have been implementations that take advantage of data structures to improve the execution time of *k*NN. One such example is the use of hashing algorithms for approximate nearest neighbor [40]. LSH (Locality-sensitive Hashing) [30,31] is an optimization technique for hashing similar items in the same bucket and has been recently used in the context of approximate *k*NN implementations. In a study focused on HAR [41], an LSH-based *k*NN implementation was able to reduce *k*NN execution time, while maintaining similar accuracy to *k*NN when the *PAMAP2* dataset [7] is considered.

Standard *k*NN implementations for HAR typically provide high classification accuracy and relatively fast execution for training but show slow classifying performance. For example, Cao et al. [43] show that a traditional *k*NN is the fastest classifier regarding training and the slowest regarding classification compared to various ML classifiers. Most research work targeting wearable devices and smartphones, and thus requiring low energy consumption, do not consider standard *k*NN implementations because of the associated heavy computations. Concerning energy consumption savings, researchers have studied and proposed trade-offs between energy savings and classification accuracy and without involving the standard *k*NN (as in [44,45]). For example, Cao et al. [46] consider the trade-off between sampling rates and features and each activity accuracy and propose a runtime activity-sensitive strategy, consisting of a hierarchical recognition scheme and decision trees, significantly reducing energy consumption.

To make *k*NN an option for HAR systems implemented using wearable devices and smartphones, some authors have proposed modifications of *k*NN. Techniques focused on prototype reduction as in [47] present significant storage requirement reductions and enhance online classification speed. For example, the Clustered KNN proposed in [24], reduces training instances according to clusters per activity, and provides faster *k*NN implementations. However, their efficiency for online learning has not been shown and is questionable because of the additional processing associated with reducing instances. Other approaches to decrease *k*NN classification time focus on search algorithms using tree-based data structures [48] and approximate nearest neighbor (ANN). Relevant examples of ANN search algorithms are based on hashing for nearest neighbor search (such as LSH [40]). Although the efficiency of these approaches has been studied in terms of execution time and classification accuracy, their analysis in terms of energy-efficiency and online learning using wearable devices and smartphones has not been done to the best of our knowledge.

Other authors target hardware accelerators and present significant performance speedups and energy-efficient solutions for both standard and approximate *k*NN implementations. For example, recent approaches using FPGAs include implementations of a standard *k*NN [49], and of a distance-based hashing approximate *k*NN [50]. They provide significant achievements, that are, however, at the moment, not possible to replicate in wearable devices and smartphones due to the typical lack of FPGA-based hardware acceleration in these devices.

Our work distinguishes from the related work, as we provide a path to *k*NN prototyping for wearable devices and smartphones by extending standard and approximate *k*NNs with substitution schemes able to manage online learning in the presence of storage and real-time requirements. The approaches proposed in this paper focus on simple Prototyping Reduction schemes without affecting too much the execution time and energy consumption.

3. *k*NN Substitution Schemes

Our HAR system obeys to the typical organization and stages considered by numerous authors. Figure 1 shows the main stages of the two views of the system: the offline training view, and the online classification view (with the possibility to include online learning).

The HAR system pipeline follows the steps: extraction of features from the sensors' raw data; normalization of those features; and then the features are used in the classification task. We note that in some HAR systems a filtering (noise reduction) step is also considered. In this work, the HAR system starts with a classifier with an offline trained model (i.e., with training instances of activities from representative users). At run-time, periods of online learning are enabled and during those periods the classifier model is incrementally updated.

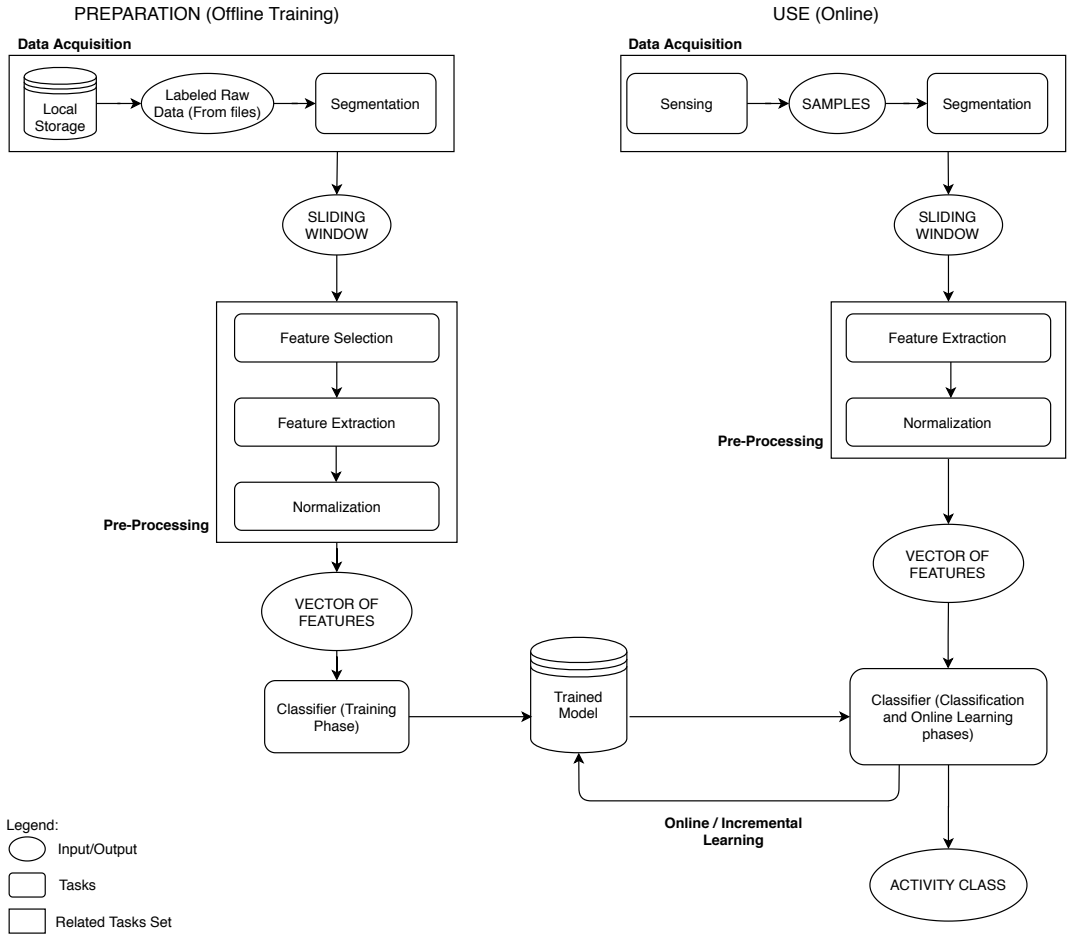


Figure 1. Block diagram of the Human Activity Recognition (HAR) system considered. In the case of *k*NN prototypes the Trained Model is a storage of instances.

The *k*NN is a lazy learner, which means that it does not have an explicit learning phase, but instead stores the training instances. For each learning instance, a vector of feature values is stored in conjunction with its label (class). Imposing a maximum limit of L, means that *k*NN can only store L instances.

The default *k*NN used already includes a substitution scheme. In this scheme (herein mentioned as "default"), whenever a new training instance needs to be considered and the maximum limit L of instances that *k*NN can store is reached, the new training instance substitutes the oldest stored instance. This scheme keeps the set of training instances updated for a user, but does not ensure a balancing model. When the number of instances that can be stored by *k*NN is limited and with the continued substitution of the oldest instance, whenever a new training instance arrives, it may happen that, due to this scheme, activities run out of training instances stored in *k*NN. This may cause classification errors when *k*NN tries to classify an instance for which it has none or insufficient training instances. Because of this, it is important to propose efficient substitution schemes able to prevent a given activity from running out of training instances stored by *k*NN.

The following eight simple, energy/computationally efficient, and real-time feasible substitution schemes are proposed, implemented, and evaluated:

- **Default:** In the *k*NN used (present in the MOA library [51]) when the instance limit is reached, whenever a new training instance arrives and needs to be stored, it replaces the oldest instance stored, regardless of its activity;
- **SS1:** randomly selects an instance of the class of the new instance and replaces it with the new instance;

- **SS2**: selects the oldest instance of the class of the new instance and replaces it with the new instance;
- **SS3:** classifies the new instance. If the new instance is incorrectly classified, SS1 is applied. Otherwise the new instance is discarded;
- **SS4:** classifies the new instance. If the new instance is incorrectly classified, SS2 is applied. Otherwise the new instance is discarded;
- **SS5**: classifies the new instance. If the new instance is correctly classified, the scheme randomly selects to apply SS1 or to discard the new instance. If the instance is classified incorrectly it applies SS1;
- **SS6**: classifies the new instance. If the new instance is correctly classified, the scheme randomly selects to apply SS2 or to discard the new instance. If the instance is classified incorrectly it applies SS2;
- **SS7**: classifies the new instance. If the new instance is incorrectly classified, the new instance replaces its nearest neighbor;
- **SS8**: classifies the new instance. If the new instance is correctly classified, the scheme randomly does one of these two options, (1) it replaces the nearest neighbor of the new instance by the new instance, or (2) it discards the new instance. If the new instance is incorrectly classified, the new instance replaces its nearest neighbor.

4. Experimental Setup

This section presents the evaluation of a prototyping HAR system consisting of *k*NN classifiers. The evaluation includes classification accuracy, execution time, and energy consumption.

4.1. Prototypes

All the evaluated HAR prototypes using *k*NN were implemented in Java and using the MOA library [51], since it allows dealing with data streams and offers a collection of incremental ML algorithms. For the LSH-based *k*NN, we use the LSH parameters empirically selected in preliminary experiments [41], i.e., 20 hashtables, 1 random projection per hash value, and 10 as the width of the projection.

Besides the execution of *k*NN prototypes in a desktop computer, we also evaluated those prototypes when running on an embedded computing board. For that, we conducted experiments in an ODROID-XU + E6 (https://www.hardkernel.com/) system running Android. The board includes a Samsung Exynos 5410 SoC, a big.LITTLE Octa-core mobile processor, with one quad-core ARM Cortex-A15 up to 1.6 GHz and one quad-core ARM Cortex-A7 up to 1.2 GHz. The ODROID-XU+E6 board includes four current/voltage sensors to measure individually the power consumption of the A15, A7, GPU and DRAM [52]. The energy consumption reported by our experiments is the product of the average power consumption (consider the A15 cores, A7 cores and DRAM) during the execution of each specific code by its execution time. We also note that since our prototype implementations of the *k*NN methods are not parallelized (e.g., using threads) only the A15 CPU and a single core was used by the *k*NN prototypes.

4.2. Datasets

We conducted several experiments with two public datasets widely used in HAR literature: *PAMAP2* (Physical Activity Monitoring for Aging People) [7,27], and *WISDM* (WIreless Sensor Data Mining) [26].

The *PAMAP2* [7,27] dataset contains 1,926,896 samples of raw sensor's data from 9 different users and 18 different activities. The data were collected from 3 devices positioned in different body areas: wrist, chest and ankle. Each device has 3 sensors embedded: a 3-axis accelerometer, a 3-axis gyroscope and a 3-axis magnetometer. The activities are organized as: basic activities (walking, running, Nordic walking and cycling); posture activities (lying, sitting and standing); everyday activities (ascending

and descending stairs); household (ironing and vacuum cleaning) and fitness activities (rope jumping). In addition, the users were encouraged to perform optional activities (watching TV, computer work, car driving, folding laundry, house cleaning and playing soccer).

The *WISDM* [26] dataset contains 5418 instances from 6 different activities. The activities present in the dataset are: Walking, Jogging, Upstairs, Downstairs, Sitting and Standing. To build the dataset, data was collected from the accelerometer of a smartphone. The data was collected with an Android smartphone carried in the front leg pocket of 20 subjects while performing the mentioned activities—[26].

4.3. Feature Extraction

In the case of the *PAMAP2* dataset, features were extracted from fixed-size sliding windows of raw data. In the experiments presented in this paper, we use a window size of 300 data samples (contiguous sensor readings) and an overlapping of 10% between consecutive sliding windows.

The PAMAP2 version of our HAR prototype used in this paper, extract 10 features for each 3D sensor: x axis Mean, y axis Mean, z axis Mean, Mean of the sum of the x, y and z axes, x axis Standard Deviation, y axis Standard Deviation, z axis Standard Deviation, x and y axes Correlation, x and z axes Correlation, and y and z axes Correlation. This results in a total of 90 features. All features are then subsequently normalized by dividing each feature value in a vector (instance) with the feature's magnitude (previously calculated using the values of each feature in the training instances).

In the case of the *WISDM* dataset, features are part of the dataset and were previously calculated from raw data. Each instance has 43 features. The features extracted from the raw data were: Average acceleration (for each axis) resulting in 3 features; Standard deviation (for each axis) resulting in 3 features; Average Absolute Difference (for each axis) resulting in 3 features; Average of the square roots of the sum of the values of each axis squared over the example duration, resulting in 1 feature; Time Between Peaks (for each axis) resulting in 3 features; and finally Binned Distribution, resulting in 30 features [26].

4.4. Dataset Sampling

In order to conduct the experiments with *PAMAP2* and *WISDM* datasets, we split the data of each dataset into a training set and a test set. Although the datasets have data collected by different users (*PAMAP2* includes data from 8 users and *WISDM* includes data from 36 users), it was not possible to use the same approach in both datasets. The low number of instances per user of the *WISDM* dataset prevents the use of the same approach used for *PAMAP2* to conduct the proposed experiments. Despite in both datasets we use a hold-out approach, this is done with differences as follows.

Regarding the *PAMAP2* dataset, user 5 was selected to represent the user of the system (i.e., the test set) and the remaining users constitute the offline training set. Being user 5 the user with the highest number of instances, it is the most suitable to carry out the proposed experiments. Regarding the *WISDM* dataset, 70% of the data was used for the training set and the remaining 30% for the test set.

Table 1 summarizes the characteristics of the datasets used.

Table 1. Characteristics of the datasets used and number of features per dataset.

Dataset	#Features	#Classes	Total Instances (#Windows)	Train Set Instances (#Windows)	Test Set Instances (#Windows)
WISDM	43	6	5418	3793	1625
PAMAP2	90	18	7194	6186	1008

4.5. Instance Limits and Instance Reduction

The number of training instances L that the kNN methods can store was defined based on the number of instances in the datasets used for the experiments. The maximum limit was set to the value that can store all instances of the dataset. Thus, the maximum limit for the *PAMAP2* dataset was set to 8000, and for the *WISDM* dataset to 6000. Other values considered for L were 100, 200, 500, 1000, and increments of 1000 until reaching 8000.

To evaluate the impact of limiting the number of instances to the kNN prototypes, it was necessary to reduce the number of instances of the training set when the prototypes cannot store all the training instances. We use stratified sampling as resampling strategy for the various limits. This way we guarantee the distribution of classes regardless of the limit of instances. The training set has been reduced starting at the 100 limit and ending at the maximum limit for each dataset. The reduction was made incrementally, i.e., except for limit 100, all limits have the instances of the immediately previous limit and then new instances are added to complete the limit. For example, for limit 1000, instances of limit 500 are kept and 500 new instances are added. The instances chosen for each limit were chosen randomly.

4.6. Experiments Description

We consider the three following scenarios and we include tests for the various storage instance limits imposed to the prototype kNN's under study, and according to the datasets used. In the time periods used for the scenarios, the instant $t0$ refers to the initial status of the kNN prototype (i.e., the one deployed and with the offline selected training instances stored) when starting the online execution.

4.6.1. Without Online Learning

In this case, each kNN prototype only uses the previously stored instances from the training set. In this experiment only *PAMAP2* was used. The test phase is done with a unique user, user 5, and the kNN prototype is not updated with the tested instances (i.e., there is not online learning).

4.6.2. With Online/Incremental Learning

The time t_0 refers to the initial state of the model. The model, for each limit, initially contains the selected instances of the training set for that limit, as explained in Section 4.5.

In this set of experiments, we consider the period $]t_0; t_1]$. In this period the initial kNN prototype can be updated, depending on the scheme used, with new instances coming from the test set. During this period, before each instance is used to incrementally update the kNN prototype, it is classified by the current kNN prototype at that particular instant. The accuracy of the kNN prototype is calculated, and then the new instance is incorporated (depending on the scheme used) into the kNN prototype. At instant t_1, the kNN prototype is updated with the instances used for online learning (i.e., the kNN prototype now reflects the dataset of the user of the HAR system, i.e., user 5 for the experiments).

4.6.3. Impact of Online Learning

In this set of experiments, we evaluate a scenario where the same instances can be used as test instance and as training instance as well. Doing this, the kNN prototype, specially without constraints, will have an accuracy of 100%, for $k = 1$ or close to 100% for other values of k. The intention of these experiments with such unrealistic scenario is to observe the impact on accuracy of both using a non-perfect kNN prototype, e.g., specially with constraints, and a sequence of instances that can reflect real-life cases, such as the fact that activities may repeat without online learning enabled. In this set of experiments we consider a period $]t_1; t_2]$ which starts with the kNN prototype updated online using the test set and then we test the model with the same dataset previously used to update the kNN prototype, but this time without online learning (i.e., no update of the training instances, and the

*k*NN prototype at instant t_2 is the same as the *k*NN prototype at t_1). The only intention is to test the predictive accuracy in an online learning setting.

5. Results and Discussion

We present herein the results achieved from the experiments we conducted for the different *k*NN prototypes.

5.1. Comparison between the kNN Methods without Online Learning

This experiment aims at analyzing the impact of limiting the number of training instances that can be stored by each *k*NN prototype on its classification accuracy. In this experiment we use the PAMAP2 dataset with the reduced training set for each limit (see Section 4.5). Then, the instances of user 5 were used as test set.

As expected, the results in Figure 2 show that as the number of instances that the *k*NN prototypes can store increase, the accuracy of the respective methods also increase, with, however, some fluctuation cases. The average accuracy is from 61.8% to 78.4% with sizes from 100 to 8000, respectively. From Figure 2, we can also see that for L $\leq$ 2000, there is a greater variation in the accuracy values, with accuracy values range from 61.7% to 74.8%. It is also possible to observe that for L $\leq$ 2000, the default *k*NN obtained better accuracy than the other methods under study. On the other hand, for L > 2000, the variation in the accuracy values is smaller. In this case the accuracy varies from 75.9% to 78.4%. For L = 8000 (limit where is possible store all instances of the PAMAP2 dataset) the *k*NNwithPAWwithADWIN method was the one with the best result, obtaining 78.4% of accuracy. Also with L = 8000, the default *k*NN obtained an accuracy of 76.9%.

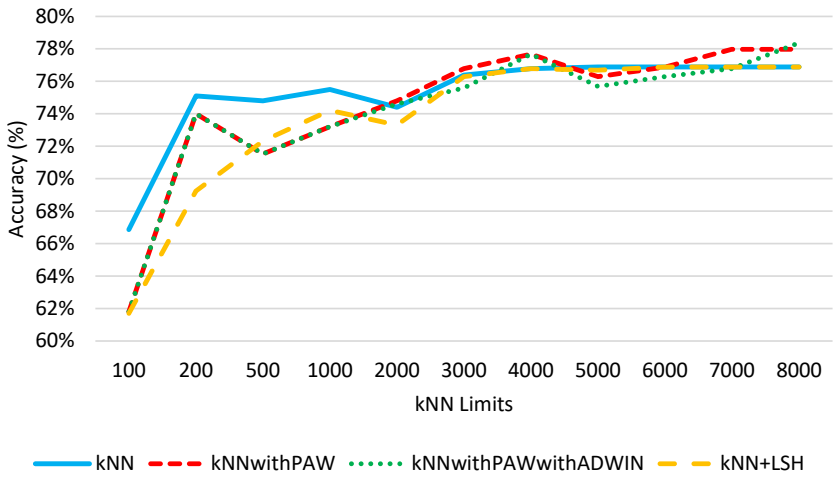


Figure 2. Impact of limiting the number of stored instances on the classification accuracy of the *k*NN methods.

5.2. Substitution Schemes Comparison

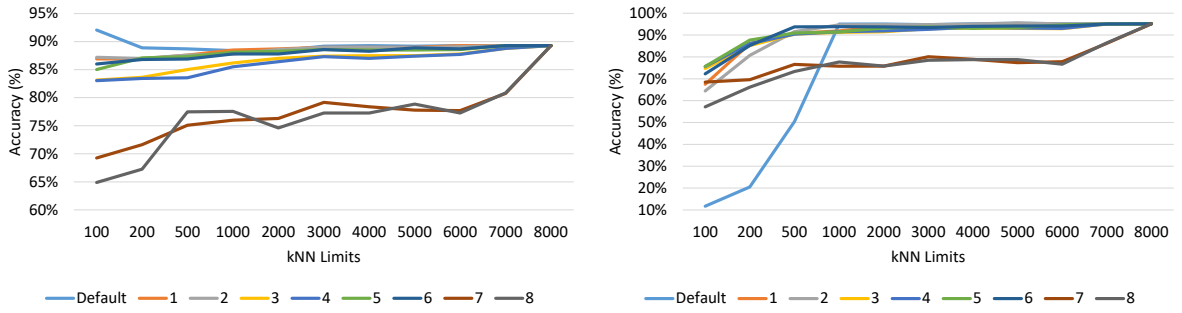
5.2.1. *PAMAP2* Dataset

In these experiments, all the substitution schemes presented in Section 3 were evaluated varying the *k*NN storage limits. Figure 3 shows the results obtained and the impact of substitution schemes on the *k*NN accuracy. In the charts of Figure 3, each line represents the accuracy of a substitution scheme for the different limits evaluated. The convergence of all the lines of the charts to the same point is due to the fact that for 8000, all the training instances are kept stored by the *k*NN prototypes. Thus, for the 8000 limit, substitutions are not applied and the accuracy is the same for all the different schemes. Comparing Figures 2 and 3b, it is possible to verify that the use of online learning improves the global

accuracy of the *k*NN prototypes. Without online learning (Figure 2) the highest accuracy is 78.4% and on the other hand with online learning (Figure 3b) the highest accuracy is 95.1%.

Figure 3a shows that, except for schemes 7 and 8, the schemes present similar results, during the incremental learning phase. The accuracy for schemes 7 and 8 is lower (from 65% to 81% when doing substitutions) than the accuracy achieved by the other schemes (from 83% to 92%), and indicate that, for the limits used in these experiments, schemes 7 and 8 seem to not be options to take into account.

As already mentioned, the set of instances used in the incremental learning phase are used again to evaluate the classification accuracy of the *k*NN prototypes after the changes made by the substitution schemes proposed in this article. Figure 3b shows that the use of the proposed substitution schemes can improve the accuracy, and are significantly better than the default substitution schemes of *k*NN, especially when the limit of instances that *k*NN can store is low ($\leq$1000). For limit 100, the use of substitution schemes allows to improve the system accuracy by approximately 56% compared to the use of the $default$ scheme. As expected, the substitution scheme of the oldest instances stored without any selection may conduct to low classification performance. This experiment also confirms that schemes 7 and 8 maintain lower accuracy when compared to the other schemes (i.e., from 1 to 6).

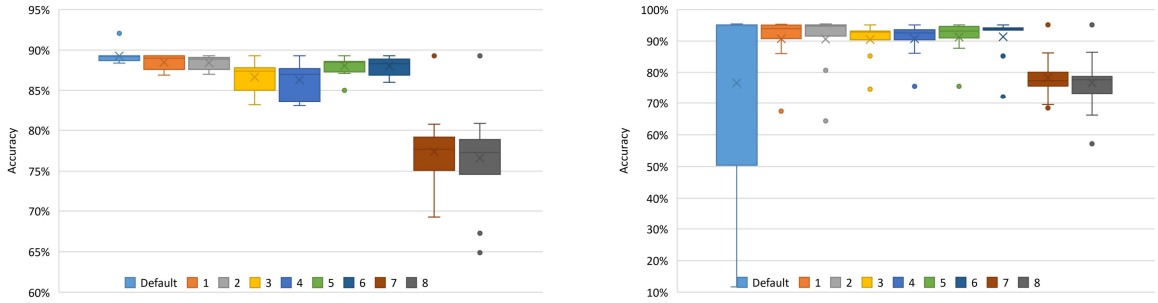


(**a**) $]t_0; t_1]$ (**b**) $]t_1; t_2]$

Figure 3. Impact on accuracy of substitution schemes for the different limits considered. (**a**) period with online learning; (**b**) period without online learning. The lines with the lowest accuracy values in graph (**b**), refer to substitution schemes 7 and 8.

Figure 4 presents the variation of accuracy for the substitution schemes evaluated. Figure 4a shows that the substitution schemes 7 and 8 had a much lower performance than the other schemes and the lower variance in the accuracy for schemes 1, 2, 5, and 6, when compared to the variances of schemes 5 and $default$.

The results in Figure 4b show that, after the Online/Incremental Learning phase, all the proposed substitution schemes obtained better results than the default *k*NN scheme and that, as we had already concluded, scheme 6 was the best among the tested schemes. The application of substitution scheme 6 during the period $]t_0; t_1]$ improved the average accuracy of the default scheme by 14.65% in the period $]t_1; t_2]$. In addition, the results in Figures 3b and 4b show that the default scheme of *k*NN obtained very low accuracy for the smallest limits (100, 200 and 500). This is due to the fact that the default scheme, when full and needing to update, always eliminates the oldest instance, leading to situations where a reduced number of activities is stored in the *k*NN prototype. In this case, the *k*NN using the default scheme conducts to an incremental update of the *k*NN prototype that in the end may represent only the latest activities used in the online training. The results also show a high variance of the $default$ scheme and similar variances for the other schemes, being scheme 6 the best option.



(**a**) $]t_0;t_1]$ (**b**) $]t_1;t_2]$

Figure 4. Performance of *k*NN with the various substitution schemes for the *PAMAP2* dataset.

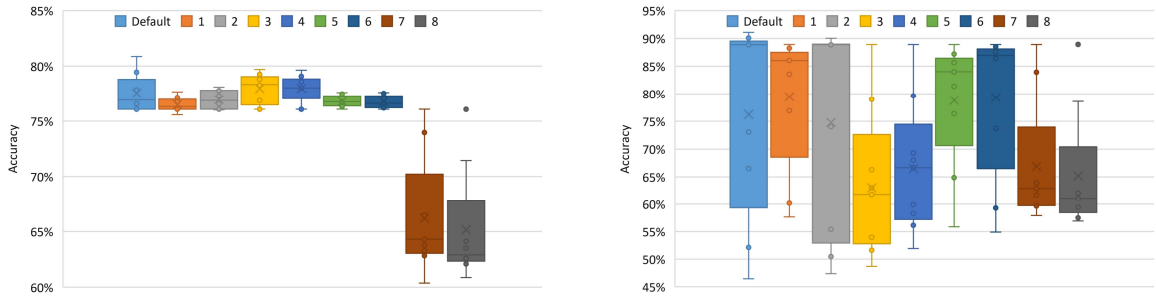
Table 2 shows the order of activities in the user 5 dataset, as well as the number of instances per activity. For the limit 100, only instances of the classes *running* and *rope_jumping* were left in the model, which means that the *k*NN prototype was only able to correctly classify instances that correspond to the identified activities. Therefore, *k*NN could only correctly classify approximately 11% of the instances of user 5, a value that has been confirmed by the results. On the other hand, our substitution schemes act at each activity group of stored instances, while traditional *k*NN methods act at the level of instances stored. Thus, using these schemes, the *k*NN prototype will always include instances of all activities in the dataset, no matter how many substitutions are made.

Table 2. Order and number of instances per activity of user 5.

Activity	**Instances**	**%**
lying	88	8.73
sitting	99	9.82
standing	82	8.13
ironing	123	12.20
vacuum_cleaning	90	8.93
ascending_stairs	53	5.26
descending_stairs	47	4.66
walking	119	11.81
nordic_walking	97	9.62
cycling	91	9.03
running	91	9.03
rope_jumping	28	2.78
Total	1008	100

5.2.2. *WISDM* Dataset

The same experiment to evaluate the Substitution Schemes for *PAMAP2* dataset was also repeated for the *WISDM* dataset. Figure 5 shows the variance in accuracy for the substitution schemes for the dataset considering the different limits used. The results show that schemes 7 and 8 had the worst accuracy results compared to the other schemes for both periods. On the other hand, schemes 1, 5 and 6 were the best schemes for this dataset. The application of substitution scheme 1 during the period $]t_0;t_1]$, improved the average accuracy of the default scheme by 3% in the period $]t_1;t_2]$.



(**a**) $]t_0;t_1]$ (**b**) $]t_1;t_2]$

Figure 5. Performance of *k*NN with the various substitution schemes for the *WISDM* dataset.

The results for $]t_1;t_2]$ in Figure 5 show that the accuracy after online learning for the default *k*NN has average values in the order of 40% for the limit size 100. These values contrast with those obtained for the *PAMAP2* dataset, where the accuracy for the default *k*NN is around 11%. One of the suspected reasons for these values is that the *WISDM* dataset has only 6 activities whereas *PAMAP2* has 12. In addition, the *PAMAP2* dataset (7194 instances) has more instances than *WISDM* dataset (5418 instances). This means that for each sequence of activities, each activity has only 10 to 20 instances. Thus, with the default *k*NN always removing the oldest instance, for limit 100, there are at least 10 instances of an activity. That is, even for the lowest limit considered and always removing the oldest instance, there always instances for most activities.

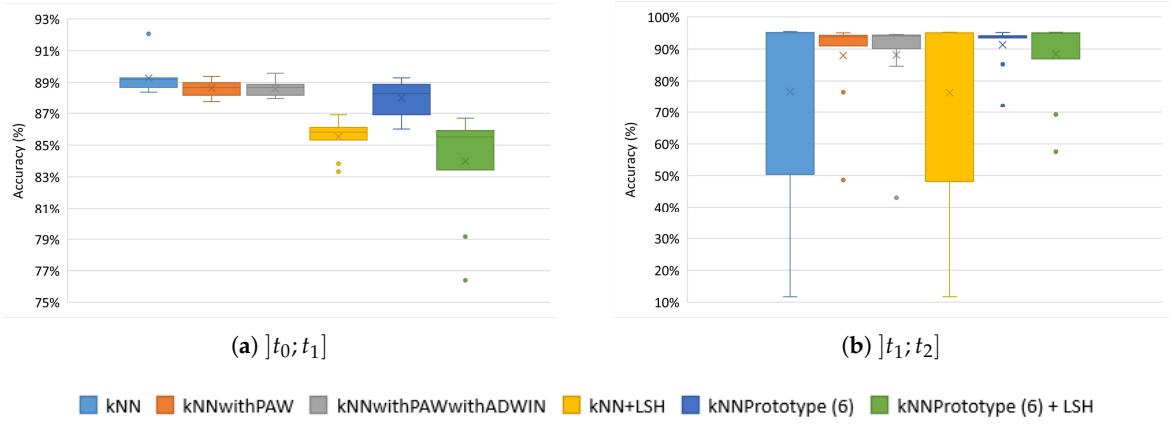
5.3. Comparison between the Various kNN Methods with Online Learning

In these experiments, we evaluated the performance of the various *k*NN methods, namely *k*NN (*default* substitution scheme), *k*NNwithPAW, *k*NNwithPAWwithADWIN, *k*NN + LSH, as well as the use of the substitution scheme 6 applied to *k*NN and to *k*NN + LSH. We selected the substitution scheme 6 in all datasets based on the results achieved for *PAMAP2*.

5.3.1. *PAMAP2* Dataset

Figure 6 shows the variance in accuracy for the *k*NN methods for the *PAMAP2* dataset. The results in Figure 6a during the Online Learning phase, show that the *k*NN method with the *default* substitution scheme achieved the best results, and in this period the proposed scheme 6 obtained inferior results, in terms of accuracy, in relation to the existing *k*NN methods. In addition, scheme 6 did not improve the *k*NN + LSH classification accuracy.

After the Online Learning phase, the method that made the best substitutions was *k*NN with our scheme 6, as shown in Figure 6b. *k*NN default was one of the worst methods, along with *k*NN default + LSH. It is also possible to verify that the *k*NNwithPAW method achieved results close to *k*NN with scheme 6, but still slightly inferior. The *k*NN and *k*NN + LSH prototypes exhibited high variances. In this experiment, however, scheme 6 improved the *k*NN + LSH classification accuracy.

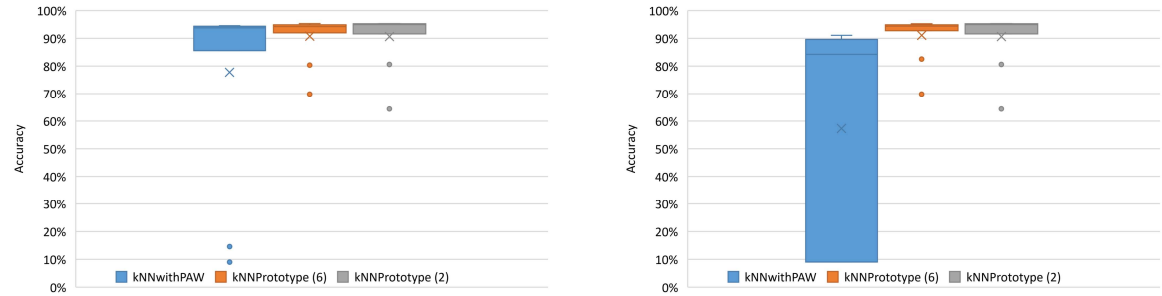


(**a**) $]t_0;t_1]$ (**b**) $]t_1;t_2]$

Figure 6. Performance of the various *k*NN methods for the *PAMAP2* dataset.

In order to study the robustness of our best methods/schemes (2 and 6) compared to the other evaluated *k*NN methods, especially *k*NNwithPAW as it had the closest results to our method, a new experiment was conducted. In this experiment, a scenario is simulated where the *k*NN prototypes are updated only with instances of an activity, in this case *running*. For this the 91 instances of *running* of user 5 were replicated $10x$ and $100x$, obtaining 910 and 9100 *running* instances, respectively. These instances were used for a new Online Learning phase after the period $]t_1;t_2]$. After this second Online Learning phase, the new *k*NN prototypes were tested using the same test set from period $]t_1;t_2]$. This experiment was performed for the methods *k*NNwithPAW, *k*NN with scheme 6 and *k*NN with scheme 2. Figure 7 shows the results obtained.

The results in Figure 7 show that our proposed substitution 2 and 6 schemes obtained better accuracy than *k*NNwithPAW, which is even more evident in Figure 7b, where 9100 *running* instances were used. These results allow us to conclude that our methods are more immune to situations where there are several instances of the same activity used to update the *k*NN prototype. Our substitution schemes act at class/activity level and in this case they substituted only instances from the *running* activity, keeping the remaining activity instances intact, which *k*NNwithPAW was unable to avoid. Thus, we can conclude that our method is more robust than the traditional and representative *k*NN methods used in this paper. Specifically, while *k*NNwithPAW presents a variation of accuracy from 9% to 90%, the accuracy for *k*NN prototype with schemes 2 and 6 only varies from 60% to 95%.



(**a**) 910 instances (**b**) 9100 instances

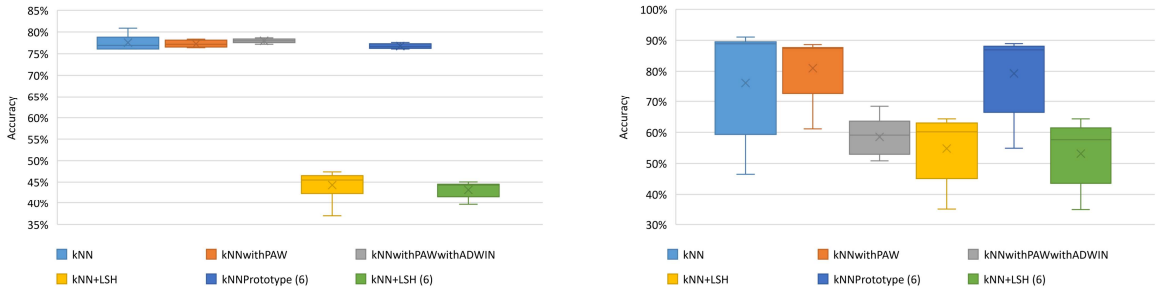
Figure 7. Performance of the *k*NN methods for the *PAMAP2* dataset after Online Learning with multiple running instances.

5.3.2. *WISDM* Dataset

The same experiment to evaluate the *k*NN methods for *PAMAP2* dataset were also repeated for the *WISDM* dataset. Figure 8 shows the variance in accuracy for the *k*NN methods for the *WISDM*

dataset. The results in Figure 8a show that the method that, in the period $]t_0;t_1]$, obtains better results is *k*NNwithPAWwithADWIN. On the other hand, the *k*NN methods with LSH, obtained results significantly inferior to the other methods. The default *k*NN, *k*NNwithPAW and *k*NN with scheme 6 had results close to *k*NNwithPAWwithADWIN.

After the online learning phase, the methods that allowed to update/store the most representative instances were *k*NNwithPAW and *k*NN with scheme 6, as shown in Figure 8b, with *k*NNwithPAW having results slightly superior to those of scheme 6. The other methods had lower accuracy than *k*NNwithPAW and *k*NN with scheme 6. Thus, and as with *PAMAP2*, *k*NNwithPAW and *k*NN with scheme 6 obtained the best results among the tested methods.



(**a**) $]t_0;t_1]$ (**b**) $]t_1;t_2]$

Figure 8. Performance of the various *k*NN methods for the *WISDM* dataset.

5.4. Execution Time and Energy Consumption

We also analyze the execution time and energy consumption for processing all the data from the *PAMAP2* dataset for all *k*NN prototypes and when executing them on an ODROID-XU + E6 (https://www.hardkernel.com/) board. The experiments focus on a single user, user 5, and the execution time and energy required to process 1008 instances. We measure the Execution Time and Energy Consumption for *k*NN with default substitution scheme, *k*NNwithPAW, *k*NNwithPAWwithADWIN, *k*NN + LSH, *k*NN with substitution scheme 6, and *k*NN + LSH with substitution scheme 6. In these experiments, we simulate the data acquisition from the sensors with readings from the files with the data.

The execution time was divided into two parts. The first part represents the time required to conclude the $]t_0;t_1]$ period for the *k*NN methods under study. The second part represents the execution time for the period $]t_1;t_2]$, which translates into the time it takes to classify the 1008 instances of user 5. The experiments were repeated 10 times and the results presented are the average of the measurements of the executions performed.

5.4.1. Execution Time

Figure 9a,b present the average execution time (per instance) for each *k*NN method.

In the $]t_0;t_1]$ period (see Figure 9a), the *k*NN prototypes classify instances and then use them to update the training instances stored. Bearing this in mind and with the achieved results, we can conclude that, on average, *k*NN + LSH prototypes are the fastest classifiers and the ones that allow the highest number of processed instances. On the other hand, the *k*NN with scheme 6 is the one that takes more time to process (note that this includes classification and incremental learning) the instances. The *k*NN with substitution scheme 2 showed results very similar to the traditional *k*NN. In this period, the *k*NN execution times ranged from 3.84 ms to 148.63 ms, while the *k*NNPrototype (2) varied between 3.19 ms and 147.78 ms, and the *k*NN Prototype (6) between 4.77 ms and 210.30 ms, with the latter method having the highest execution time between the studied methods. On the other hand, *k*NN + LSH varied between 0.49 ms and 11.11 ms, and *k*NN + LSH (6)

varied between 0.52 ms and 15.99 ms. *k*NN + LSH obtained the shortest execution times among the studied methods. For example, *k*NN + LSH obtained, on average, execution times 14× lower than *k*NNwithPAWwithADWIN, whose values varied between 7.56 ms and 169.40 ms, and 10× lower than *k*NNwithPAW, which varied between 5.12 ms and 116.57 ms.

On the other hand, in period $]t_1; t_2]$ (see Figure 9b), where the *k*NN prototypes only classify instances (i.e., online learning is disabled), the *k*NN + LSH prototypes were the fastest to complete all the classifications, with execution times varying between 0.35 ms and 11.01 ms and significantly shorter than the other methods under study. For example, *k*NN + LSH needed, on average, execution times 13× lower than *k*NN and 10× lower than *k*NNwithPAW. On the other hand, *k*NN was the slowest to classify each instance, with execution times varying between 3.80 ms and 159.25 ms. *k*NNwithPAW and *k*NNwithPAWwithADWIN achieved similar execution times, but much higher than the *k*NN + LSH prototypes. The execution times varied between 4.35 ms and 117.39 ms, and between 4.37 ms and 129.89 ms, for *k*NNwithPAWwithADWIN and *k*NNwithPAW, respectively. Our substitution schemes have no influence during period $]t_1; t_2]$, since this period only concerns to classification, which, in practice, results in executing *k*NN and *k*NN + LSH.

Taking into account its use in real-life, *k*NN + LSH (6) proves to be the best classifier to use on a smartphone from the point of view of the execution time and energy consumption. It is worth noting that the presented execution times neither include the data collection from the sensors nor the extraction of features from sensing data, but those steps are the same for all the *k*NN prototypes evaluated.

The sampling frequency of 100 Hz, a sliding window size of 300 samples, and an overlap of 10% between two consecutive windows, used in the experiments, impose a maximum of 2.7 s to perform feature extraction, normalization and classification. The execution times of the *k*NN methods for the Online Learning phase are between 0.49 ms for the method *k*NN + LSH, and 210.30 ms for the method *k*NNPrototype (6). For the classification phase, the execution times are between 0.35 ms for the method *k*NN + LSH and 159.25 ms for the method *k*NN. These execution times are below the time constraint of 2.7 s and would allow the system to run using the selected characteristics (i.e., sampling frequency of 100 Hz, a sliding window size of 300 samples, and an overlap of 10%).

5.4.2. Energy Consumption

Figure 9c,d show the results regarding energy consumption measurements. The results in Figure 9c show that *k*NNwithPAWwithADWIN consume more energy than the other methods. The remaining methods present lower energy consumption, with emphasis on *k*NN + LSH prototypes, which despite being the ones that most varies, are the ones with the lowest average energy consumption. For example, *k*NN + LSH, on average, consumes 23% less than *k*NNwithPAWwithADWIN.

Of the methods with the lowest energy consumption, *k*NN has the highest average value, despite having less variability in values. Schemes 2 and 6 have energy consumption similar to *k*NN, but consume slightly more energy than *k*NNwithPAW. Comparing our schemes, scheme 2 presents an energy consumption below scheme 6.

The results in Figure 9d show that the *k*NNwithPAWwithADWIN method is the one consuming more energy, the *k*NN + LSH prototypes are the ones with the lowest minimum and average energy consumption, and the other methods exhibit similar energy consumption. *k*NN + LSH, on average, consumes 20% less than *k*NNwithPAWwithADWIN.

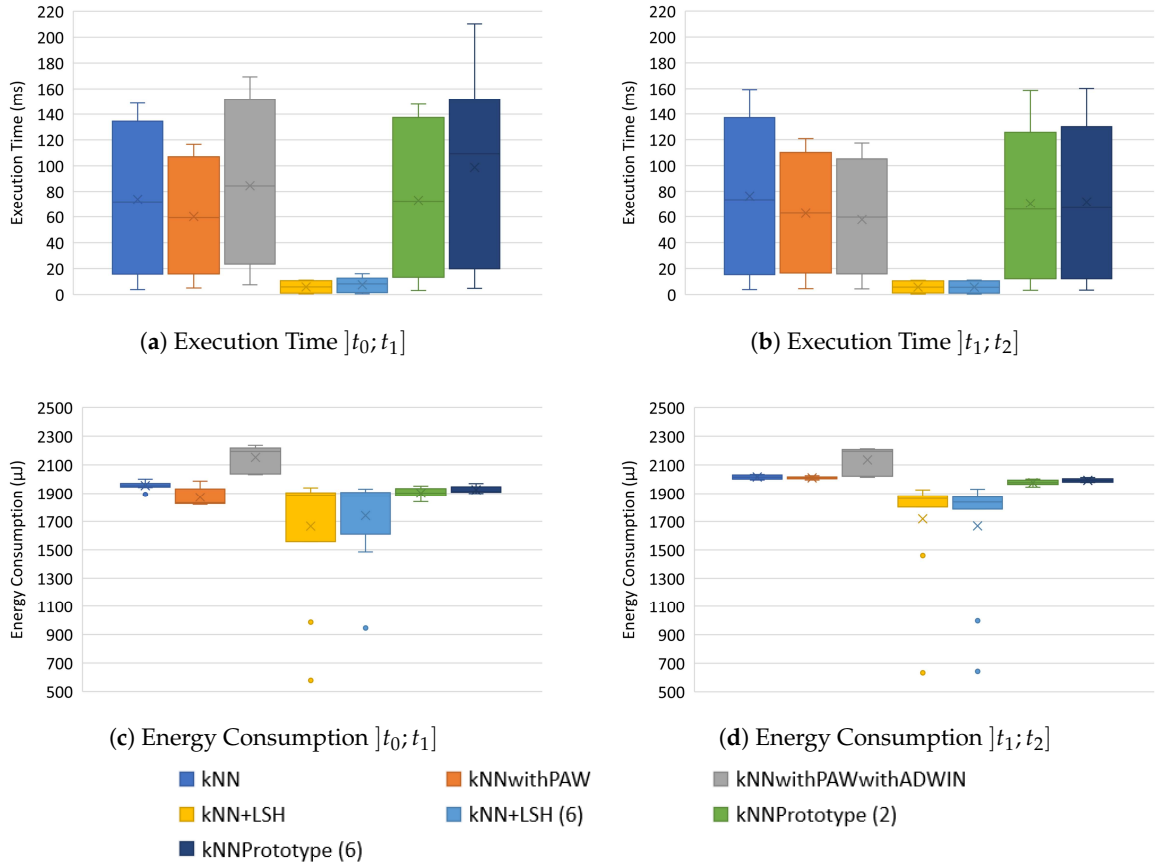


Figure 9. Energy Consumption (µJ) and Execution Time (ms) (per instance) of the various *k*NN methods for the *PAMAP2* dataset.

5.5. Analysis of Results

Previous work on HAR addressing smartphones, present high accuracy for HAR when using *k*NN, see, e.g., Morillo et al. [53]. The best approaches presented in this paper can maintain the standard *k*NN human-activity recognition accuracy while dealing simultaneously with typical constraints present on embedded systems (e.g., wearable devices) and low energy consumption increases. However, the accuracy achieved will always depend on the maximum number of training instances stored and substitution policies when using online learning.

The prototype reduction techniques used offline are essential to deploy efficient *k*NN-based HAR systems. At runtime, the substitution policies are of paramount importance because of their capability to update the instances stored with more representative ones but need to deal with very stringent and strict requirements. Previous work has avoided the use of *k*NN (considering it offline and/or online learning) in the context of wearable devices due to the constraints of memory on those devices, see, e.g., [24], and in the context of online learning, e.g., [54]. The authors are thus forced to use other techniques, in most cases with higher online learning complexity, and without analyzing the impact of *k*NN. Our results suggest that our approaches for maintaining a maximum size of search space and thus respecting maximum storage requirements are steps for the use of *k*NN in embedded systems and also making possible the use of online learning in real situations.

The results presented in this paper show some of the capability of simple approaches and their alignment to make *k*NN a solution for online learning and make evident that there is room to improve the current approaches, both in terms of accuracy, execution time and energy savings.

6. Conclusions

Although *k*NN is one of the most analyzed classifiers for Human Activity Recognition (HAR) systems, its storage and computing requirements increase with the number of features and training instances and thus bring additional prototyping challenges. This paper proposed schemes to update/substitute the set of instances for *k*NN classifiers in order to maintain a maximum number of stored training instances when considering Online/Incremental Learning. The eight proposed schemes are evaluated in the context of a HAR prototyping system.

According to the experimental results, simple substitution schemes can be very useful for embedded HAR system implementations in devices with limited memory (e.g., wearable devices). The results show the efficiency of some of the eight schemes evaluated in two datasets widely used in HAR literature, both in terms of accuracy, execution time, and energy consumption.

In addition, we evaluated additional *k*NN prototypes and analyzed their behavioral considering different aspects that mimic real-life scenarios. The results show that, although the simple substitution schemes impose execution time and energy consumption overhead to typical *k*NN prototypes, their robustness with respect to some scenarios justify their adoption.

Ongoing work is focused on additional experiments with scenarios including larger number of activities and sensing data. As future work, we plan to continue researching substitution schemes and compare it with other more computational intensive schemes and verify the overheads regarding response time and energy consumption. We have also plans to investigate the possibility of dynamic adaptation of the number of training instances per class.

Author Contributions: Conceptualization, Formal analysis, Funding acquisition, Visualization, and Supervision: J.M.P.C. and J.M.-M.; Resources, Software, and Writing—original draft: P.J.S.F.; Investigation, and Writing—review & editing: J.M.P.C. and J.M.-M. Investigation, P.J.S.F. and J.M.P.C.; Data curation: P.J.S.F., J.M.P.C. and J.M.-M.; Methodology, and Validation: P.J.S.F. and J.M.-M.; Project administration, J.M.P.C. All authors have read and agreed to the published version of the manuscript.

Funding: This work has been partially funded by the European Regional Development Fund (ERDF) through the Operational Programme for Competitiveness and Internationalisation—COMPETE 2020 Programme and by National Funds through the Fundação para a Ciência e a Tecnologia (FCT) within project POCI-01-0145-FEDER-016883.

Acknowledgments: We thank to R.M.C. Magalhães for the first implementation of the baseline *k*NN + LSH prototype, and to K.D. Garcia for her support regarding *k*NN prototype reduction techniques using clustering.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

HAR	Human Activity Recognition
*k*NN	*k*-Nearest Neighbour
ML	Machine Learning
LSH	Locality-Sensitive Hashing

References

1. Lara, O.D.; Labrador, M.A. A Survey on Human Activity Recognition using Wearable Sensors. *IEEE Commun. Surv. Tutorials* **2013**, *15*, 1192–1209. [CrossRef]
2. Shoaib, M.; Bosch, S.; Incel, O.D.; Scholten, H.; Havinga, P.J. A Survey of Online Activity Recognition Using Mobile Phones. *Sensors* **2015**, *15*, 2059–2085. [CrossRef]
3. Shoaib, M.; Bosch, S.; Incel, O.; Scholten, H.; Havinga, P. Complex human activity recognition using smartphone and wrist-worn motion sensors. *Sensors* **2016**, *16*, 426. [CrossRef]
4. Chen, Y.; Shen, C. Performance analysis of smartphone-sensor behavior for human activity recognition. *IEEE Access* **2017**, *5*, 3095–3110. [CrossRef]

5. Dobbins, C.; Rawassizadeh, R.; Momeni, E. Detecting physical activity within lifelogs towards preventing obesity and aiding ambient assisted living. *Neurocomputing* **2017**, *230*, 110–132. [CrossRef]
6. Sang, V.N.T.; Thang, N.D.; Van Toi, V.; Hoang, N.D.; Khoa, T.Q.D. Human Activity Recognition and Monitoring Using Smartphones. In Proceedings of the 5th International Conference on Biomedical Engineering, Ho Chi Minh, Vietnam, 16–18 June 2014; pp. 481–485.
7. Reiss, A.; Stricker, D. Introducing a New Benchmarked Dataset for Activity Monitoring. In Proceedings of the 16th IEEE International Symposium on Wearable Computers (ISWC), Newcastle UK, 18–22 June 2012; pp. 108–109. [CrossRef]
8. Banos, O.; Garcia, R.; Holgado-Terriza, J.A.; Damas, M.; Pomares, H.; Rojas, I.; Saez, A.; Villalonga, C. mHealthDroid: A novel framework for agile development of mobile health applications. In *International Workshop on Ambient Assisted Living*; Springer: Belfast, UK, 2014; pp. 91–98.
9. Lara, O.D.; Labrador, M.A. A mobile platform for real-time human activity recognition. In Proceedings of the 2012 IEEE Consumer Communications and Networking Conference (CCNC), Las Vegas, NV, USA, 14–17 January 2012; pp. 667–671.
10. Liang, Y.; Zhou, X.; Yu, Z.; Guo, B.; Yang, Y. Energy Efficient Activity Recognition Based on Low Resolution Accelerometer in Smart Phones. In *Advances in Grid and Pervasive Computing*; Li, R., Cao, J., Bourgeois, J., Eds.; Springer: Berlin/Heidelberg, Germany, 2012; pp. 122–136.
11. Siirtola, P.; Röning, J. Ready-to-use activity recognition for smartphones. In Proceedings of the 2013 IEEE Symposium on Computational Intelligence and Data Mining (CIDM), Singapore, 16–19 April 2013; pp. 59–64.
12. Martín, H.; Barbolla, A.; Iglesias, J.; Casar, J. Activity Logging Using Lightweight Classification Techniques in Mobile Devices. *Pers. Ubiquitous Comput.* **2013**, *17*, 675–695. [CrossRef]
13. Das, B.; Seelye, A.M.; Thomas, B.L.; Cook, D.J.; Holder, L.B.; Schmitter-Edgecombe, M. Using smart phones for context-aware prompting in smart environments. In Proceedings of the 2012 IEEE Consumer Communications and Networking Conference (CCNC), Las Vegas, NV, USA, 14–17 January 2012; pp. 399–403.
14. Gomes, J.B.; Krishnaswamy, S.; Gaber, M.M.; Sousa, P.A.C.; Menasalvas, E. MARS: A Personalised Mobile Activity Recognition System. In Proceedings of the 2012 IEEE 13th International Conference on Mobile Data Management, Bengaluru, India, 23–26 July 2012; pp. 316–319.
15. Mannini, A.; Intille, S.S.; Rosenberger, M.; Sabatini, A.M.; Haskell, W. Activity recognition using a single accelerometer placed at the wrist or ankle. *Med. Sci. Sports Exerc.* **2013**, *45*, 2193. [CrossRef]
16. Siirtola, P.; Röning, J. Recognizing Human Activities User-independently on Smartphones Based on Accelerometer Data. *Int. J. Interact. Multimed. Artif. Intell.* **2012**, *1*, 38–45. [CrossRef]
17. Xu, C.; Chai, D.; He, J.; Zhang, X.; Duan, S. InnoHAR: A Deep Neural Network for Complex Human Activity Recognition. *IEEE Access* **2019**, *7*, 9893–9902. [CrossRef]
18. Ignatov, A. Real-time human activity recognition from accelerometer data using Convolutional Neural Networks. *Appl. Soft Comput.* **2018**, *62*, 915–922. [CrossRef]
19. Ronao, C.A.; Cho, S.B. Human activity recognition with smartphone sensors using deep learning neural networks. *Expert Syst. Appl.* **2016**, *59*, 235–244. [CrossRef]
20. Wan, S.; Qi, L.; Xu, X.; Tong, C.; Gu, Z. Deep learning models for real-time human activity recognition with smartphones. *Mob. Netw. Appl.* **2020**, *25*, 743–755. [CrossRef]
21. Hassan, M.; Uddin, M.Z.; Almogren, A. A robust human activity recognition system using smartphone sensors and deep learning. *Future Gener. Comput. Syst.* **2017**, *81*, 307–313. [CrossRef]
22. Mohamad, S.; Sayed-Mouchaweh, M.; Bouchachia, A. Online active learning for human activity recognition from sensory data streams. *Neurocomputing* **2020**, *390*, 341–358. [CrossRef]
23. Dearo Garcia, K.; de Carvalho, A.; Mendes-Moreira, J. A cluster based prototype reduction for online classification. In *Intelligent Data Engineering and Automated Learning—IDEAL 2018*; Yin, H., Camacho, D., Novais, P., Tallón-Ballesteros, A., Eds.; Lecture Notes in Computer Science; Springer: Madrid, Spain, 21–23 November 2018; pp. 603–610. [CrossRef]
24. Bhat, G.; Deb, R.; Chaurasia, V.V.; Shill, H.; Ogras, U.Y. Online Human Activity Recognition using Low-Power Wearable Devices. In Proceedings of the 2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), San Diego, CA, USA, 5–8 November 2018; pp. 1–8.

25. Ferreira, P.J.S.; Magalhães, R.M.C.; Garcia, K.D.; Cardoso, J.M.P.; Mendes-Moreira, J. An Efficient Scheme for Prototyping kNN in the Context of Real-Time Human Activity Recognition. In *Intelligent Data Engineering and Automated Learning–IDEAL 2019*; Yin, H., Camacho, D., Tino, P., Tallón-Ballesteros, A.J., Menezes, R., Allmendinger, R., Eds.; Springer International Publishing: Cham, Switzerland, 2019; pp. 486–493.
26. Kwapisz, J.R.; Weiss, G.M.; Moore, S.A. Activity recognition using cell phone accelerometers. In Proceedings of the Fourth International Workshop on Knowledge Discovery from Sensor Data, Washington, DC, USA, 4–30 May 2010; pp. 10–18.
27. Weiss, G.M.; Yoneda, K.; Hayajneh, T. Smartphone and Smartwatch-Based Biometrics Using Activities of Daily Living. *IEEE Access* **2019**, *7*, 133190–133202. [CrossRef]
28. Bifet, A.; Pfahringer, B.; Read, J.; Holmes, G. Efficient Data Stream Classification via Probabilistic Adaptive Windows. In Proceedings of the 28th Annual ACM Symposium on Applied Computing, Coimbra, Portugal, 18–22 March 2013; pp. 801–806. [CrossRef]
29. Bifet, A.; Gavalda, R. Learning from time-changing data with adaptive windowing. In Proceedings of the 2007 SIAM International Conference on Data Mining, Minneapolis, Minnesota, 26–28 April 2007; Volume 7, pp. 443–448.
30. Indyk, P.; Motwani, R. Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality. In Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing, Dallas, TX, USA, 23–26 May 1998; pp. 604–613. [CrossRef]
31. Datar, M.; Immorlica, N.; Indyk, P.; Mirrokni, V.S. Locality-sensitive hashing scheme based on p-stable distributions. In Proceedings of the 20th Annual Symposium on Computational Geometry—SCG'04, Brooklyn, NY, USA, 9–11 June 2004; p. 253. [CrossRef]
32. Nanni, L.; Lumini, A. Prototype reduction techniques: A comparison among different approaches. *Expert Syst. Appl.* **2011**, *38*, 11820–11828. [CrossRef]
33. Garcia, S.; Derrac, J.; Cano, J.; Herrera, F. Prototype Selection for Nearest Neighbor Classification: Taxonomy and Empirical Study. *IEEE Trans. Pattern Anal. Mach. Intell.* **2012**, *34*, 417–435. [CrossRef]
34. Triguero, I.; Derrac, J.; Garcia, S.; Herrera, F. A Taxonomy and Experimental Study on Prototype Generation for Nearest Neighbor Classification. *IEEE Trans. Syst. Man Cybern. Part C Appl. Rev.* **2012**, *42*, 86–100. [CrossRef]
35. Hart, P. The condensed nearest neighbor rule (Corresp.). *IEEE Trans. Inf. Theory* **1968**, *14*, 515–516. [CrossRef]
36. Kira, K.; Rendell, L.A. The Feature Selection Problem: Traditional Methods and a New Algorithm. In Proceedings of the Tenth National Conference on Artificial Intelligence, San Jose, CA, USA, 12–16 July 1992; pp. 129–134.
37. Gallego, A.J.; Calvo-Zaragoza, J.; Valero-Mas, J.J.; Rico-Juan, J.R. Clustering-based k-nearest neighbor classification for large-scale data with neural codes representation. *Pattern Recognit.* **2018**, *74*, 531–543. [CrossRef]
38. Athitsos, V.; Potamias, M.; Papapetrou, P.; Kollios, G. Nearest Neighbor Retrieval Using Distance-Based Hashing. In Proceedings of the 2008 IEEE 24th International Conference on Data Engineering, Cancun, Mexico, 7–12 April 2008; pp. 327–336. [CrossRef]
39. He, J.; Chang, S.; Radhakrishnan, R.; Bauer, C. Compact hashing with joint optimization of search accuracy and time. In Proceedings of the CVPR, Colorado Springs, CO, USA, 20–25 June 2011; pp. 753–760. [CrossRef]
40. Andoni, A.; Indyk, P. Near-Optimal Hashing Algorithms for Approximate Nearest Neighbor in High Dimensions. *Commun. ACM* **2008**, *51*, 117–122. [CrossRef]
41. Magalhães, R.M.C.; Cardoso, J.M.P.; Mendes-Moreira, J. Energy Efficient Smartphone-Based Users Activity Classification. In Proceedings of the 19th EPIA Conference on Artificial Intelligence, Vila Real, Portugal, 3–6 September 2019; Springer: Cham, Switzerland, 2019. [CrossRef]
42. Losing, V.; Hammer, B.; Wersing, H. KNN Classifier with Self Adjusting Memory for Heterogeneous Concept Drift. In Proceedings of the 2016 IEEE 16th International Conference on Data Mining (ICDM), Barcelona, Spain, 12–15 December 2016; pp. 291–300. [CrossRef]
43. Cao, L.; Wang, Y.; Zhang, B.; Jin, Q.; Vasilakos, A.V. GCHAR: An efficient Group-based Context—Aware human activity recognition on smartphone. *J. Parallel Distrib. Comput.* **2018**, *118*, 67–80. [CrossRef]
44. Zheng, L.; Wu, D.; Ruan, X.; Weng, S.; Peng, A.; Tang, B.; Lu, H.; Shi, H.; Zheng, H. A Novel Energy-Efficient Approach for Human Activity Recognition. *Sensors* **2017**, *17*, 2064. [CrossRef]

45. Yan, Z.; Subbaraju, V.; Chakraborty, D.; Misra, A.; Aberer, K. Energy-Efficient Continuous Activity Recognition on Mobile Phones: An Activity-Adaptive Approach. In Proceedings of the 2012 16th International Symposium on Wearable Computers, Newcastle, UK, 18–22 June 2012; pp. 17–24. [CrossRef]
46. Liang, Y.; Zhou, X.; Yu, Z.; Guo, B. Energy-Efficient Motion Related Activity Recognition on Mobile Devices for Pervasive Healthcare. *Mob. Netw. Appl.* **2014**, *19*, 303–317. [CrossRef]
47. Yang, T.; Cao, L.; Zhang, C. A Novel Prototype Reduction Method for the K-Nearest Neighbor Algorithm with K $\geq$ 1. In Proceedings of the 14th Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining—Volume Part II, Hyderabad, India, 21–24 June 20105; pp. 89–100. [CrossRef]
48. Muja, M.; Lowe, D.G. Scalable Nearest Neighbor Algorithms for High Dimensional Data. *IEEE Trans. Pattern Anal. Mach. Intell.* **2014**, *36*, 2227–2240. [CrossRef]
49. Vieira, J.; Duarte, R.P.; Neto, H.C. kNN-STUFF: kNN STreaming Unit for Fpgas. *IEEE Access* **2019**, *7*, 170864–170877. [CrossRef]
50. Ito, T.; Itotani, Y.; Wakabayashi, S.; Nagayama, S.; Inagi, M. A Nearest Neighbor Search Engine Using Distance-Based Hashing. In Proceedings of the 2018 International Conference on Field-Programmable Technology (FPT), Naha Okinawa, Japan, 11–15 December 2018; pp. 150–157. [CrossRef]
51. Bifet, A.; Holmes, G.; Kirkby, R.; Pfahringer, B. MOA: Massive Online Analysis. *J. Mach. Learn. Res.* **2010**, *11*, 1601–1604.
52. Baek, I.H.; Liu, X. *Power and Energy Analysis on Odroid-XU+ E and Adaptive Power Model*; University of California Los Angeles: Los Angeles, CA, USA, 2017.
53. Soria Morillo, L.; Gonzalez-Abril, L.; Ortega, J.; Álvarez de la Concepción, M. Low Energy Physical Activity Recognition System on Smartphones. *Sensors* **2015**, *15*, 5163–5196. [CrossRef]
54. Youssef, A.; Aerts, J.; Vanrumste, B.; Luca, S. A Localised Learning Approach Applied to Human Activity Recognition. *IEEE Intell. Syst.* **2020**. [CrossRef]